AIR UNIVERSITY

(Complex Engineering Activity)

(Different Project Ideas Implementation using OOP in C++)

Ramish Faraz	**(230660)**
M.Zohaib	**(230602)**
Kamran Nasar	**(230674)**

“Project Submitted To”

Dr. Mumajjed UL Mudassir

OOP PROJECT

INTRODUCTION:

In the name of Allah, **the Most Gracious, the Most Merciful,** we begin our project on Object-**Oriented Programming (OOP).** This project aims to apply the principles of OOP to solve real-world problems and enhance our understanding of software design and development. Below are **the key topics covered in this project.**

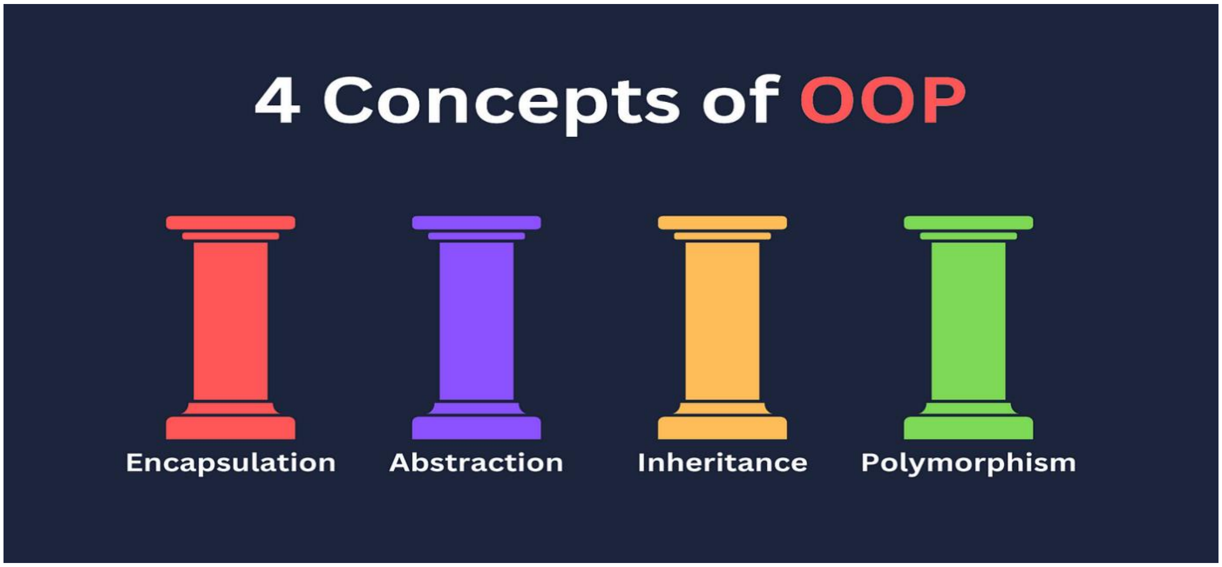
TOPICS:

- TIC-TAC-TOE GAME
- SPACE WAR GAME
- ONLINE QUIZ GAME
- UML

What is Object-Oriented Programming?

Object-Oriented Programming (OOP) in C++ is a programming paradigm that uses objects and classes to structure and manage code in a more **efficient and modular way**. C++ is a powerful language that supports OOP concepts, allowing developers to create complex and scalable software systems.

By leveraging OOP principles, C++ enables developers to write more modular, reusable, and maintainable code, making it easier to manage large and complex software projects.



Space Wars: A Case Study on the Development of a 2D Shooter Game with SFML and C++

Introduction:

"Space Wars" is a **classic 2D shooter** game where players control a spaceship and battle against waves of enemy ships. Set in the vastness of space, the game offers fast-paced action and exciting gameplay.

Background:

1. Empty Space
2. Planets
3. Stars

Development Process:

Conceptualization and Planning:

- Define the Game Concept: To tell the core idea and theme of the game. Decide on the setting (space), gameplay mechanics (shooting, dodging), and overall style (retro, futuristic, funky).
- Create a Game Design Document (GDD): Document all aspects of the game, including gameplay mechanics, level designs, art style, audio elements, and technical requirements.

Art and Asset Creation:

- Design Visual Assets: Create sprites for player ships, enemies, projectiles, backgrounds, and other visual elements. Ensure consistency in art style and resolution.Design Audio Assets: Create sound effects for actions such as shooting, explosions, and power-up pickups. Compose background music that fits the game's theme and enhances the player's experience.

Programming:

- Choose a Game Engine: Select a suitable game engine or framework for 2D game development. Popular choices include Unity with 2D Toolkit, Godot Engine, or custom frameworks using **libraries like SFML** or SDL. Here we have used SFML.
- Implement Gameplay Mechanics: Write code to handle player input, movement, shooting, enemy behavior, collision detection, and scoring. Ensure code is modular and well-structured for easier maintenance and iteration.
- Integrate Art and Audio: Integrate visual and audio assets into the game, linking animations, sprites, and sound effects to gameplay events and actions.

Level Design:

- Design Levels: Create level layouts with increasing complexity and challenge. Place enemy spawn points, obstacles, power-ups, and environmental hazards strategically to provide engaging gameplay.
- Balance Difficulty: Playtest levels to ensure a balanced difficulty curve, adjusting enemy strength, spawn rates, and environmental factors as needed.

Technologies Used:

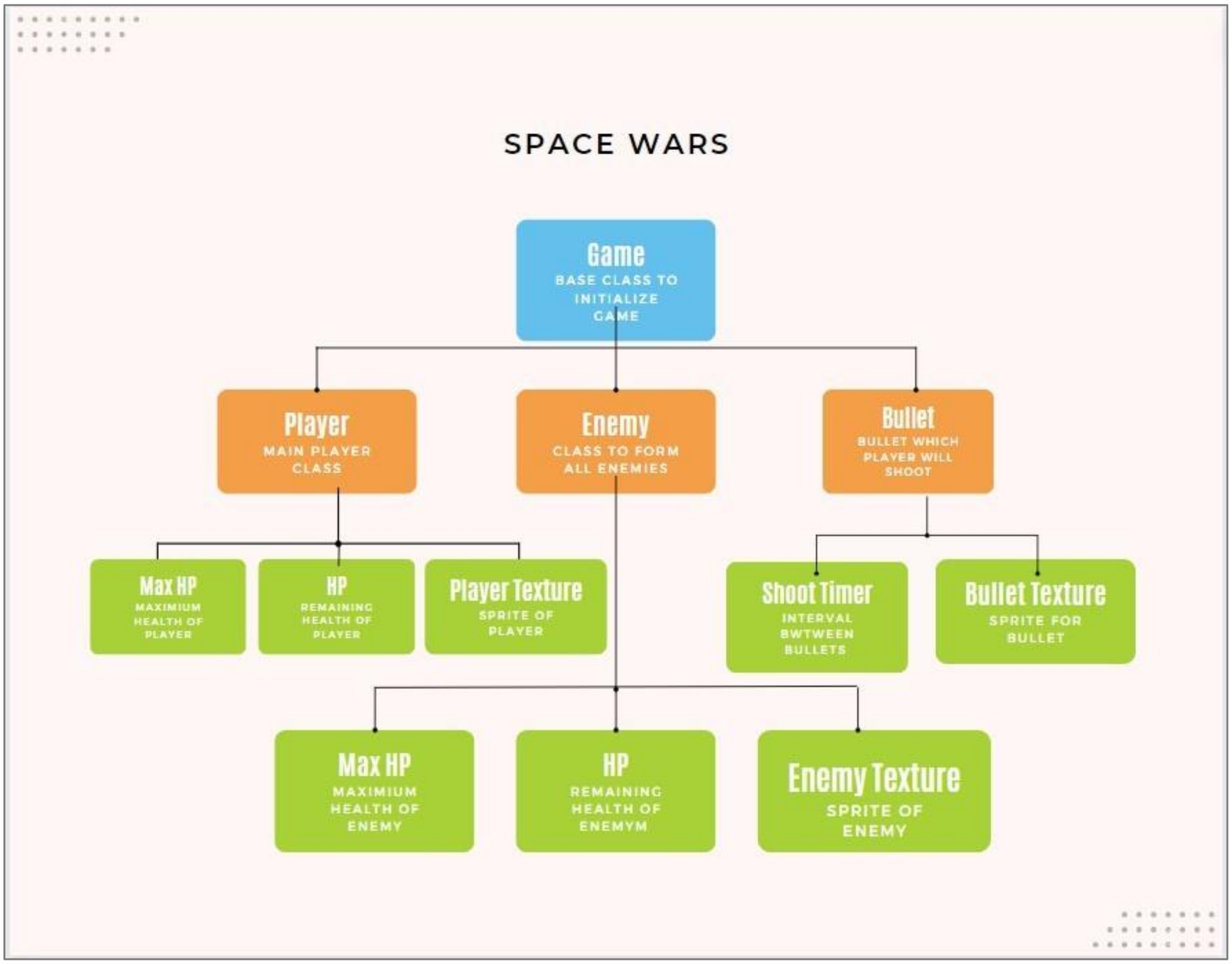
1)C++

2)SMFL

Lessons Learned:

1. **Game Design Iteration:** Iteration is crucial in game design. Testing different gameplay mechanics, enemy behaviors, and level designs allows for the refinement of the player experience.
2. **Balancing Difficulty:** Balancing the game's difficulty curve is essential to keep players engaged without overwhelming them. Gradually ramp up the challenge as player progress, offering a mix of easy, medium, and hard levels or enemy encounters.
3. **Polish and Presentation:** The overall presentation of the game greatly influences player perception. Investing time in high-quality artwork, animations, sound effects, and user interface design can elevate the game's appeal and immersion.

UML-



Conclusion: In this case study, we delved into the development process of a space shooter game using C++ and SFML. Beginning with an overview of the project goals, we proceeded to discuss the implementation details, including game mechanics, graphics rendering, player input, and sound effects.One of the notable aspects of this project was the utilization of C++ for its performance and flexibility, coupled with SFML for its simplicity and efficiency in handling multimedia components. This combination allowed for the creation of a compelling gaming experience while maintaining a manageable codebase. Throughout the development process, several challenges were encountered and addressed, such as optimizing performance, managing game state transitions, and implementing collision detection. These challenges were overcome through careful planning, iterative development.

TIC-TAC-TOE

INTRODUCTION: This report presents the development **of a Tic-Tac-Toe game** implemented in C++ using Object-Oriented Programming (OOP) principles. Tic-Tac-Toe is a classic two-player game where players take turns marking spaces on a 3x3 grid. The objective is to be the first player to place three of their marks in a horizontal, vertical, or diagonal row.

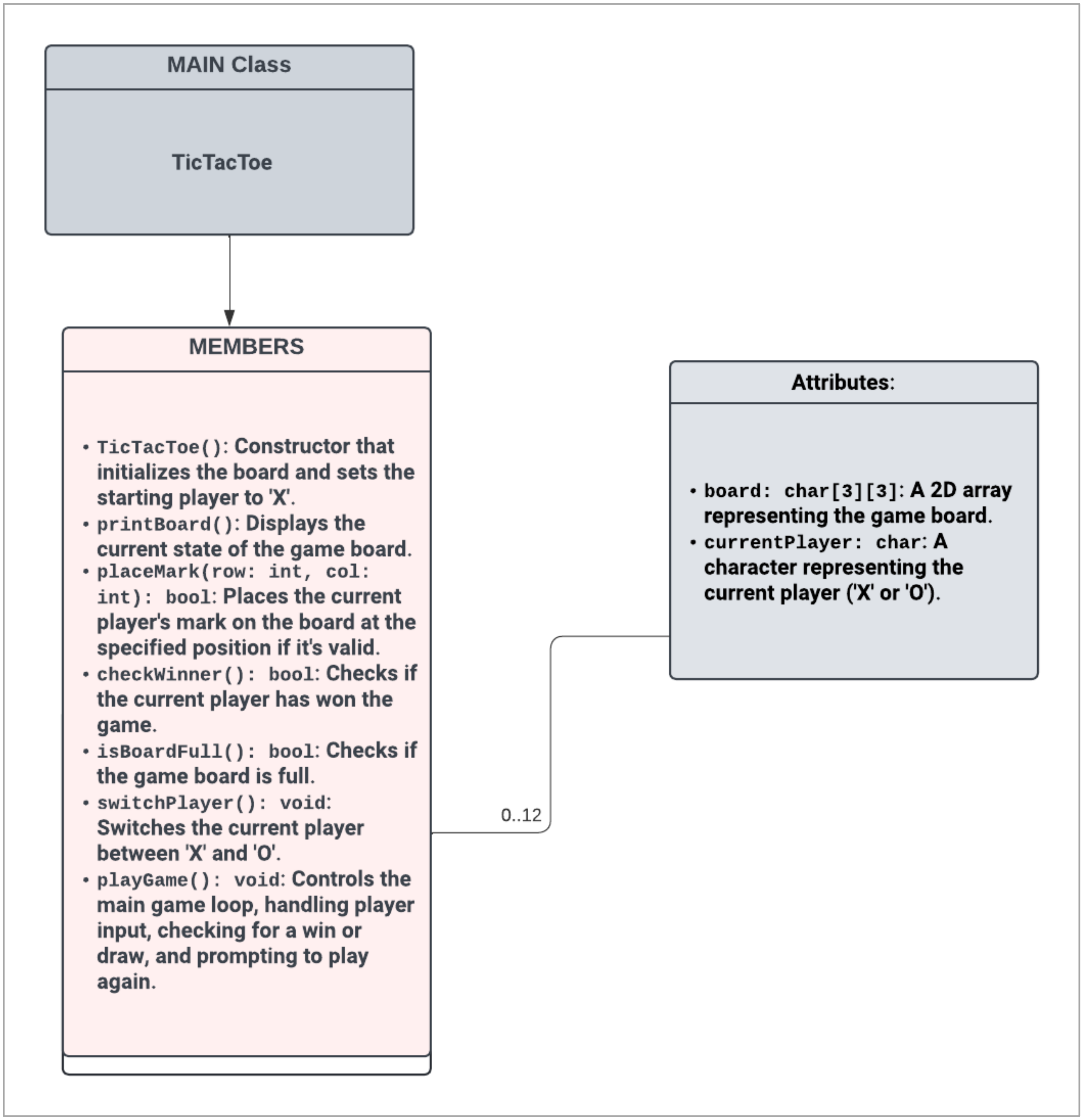
The project aims to demonstrate the practical application of OOP concepts such as **encapsulation, inheritance, and polymorphism** in C++. By structuring the game around a class that encapsulates the game state and logic, the code becomes more modular, reusable, and easier to maintain. This approach also illustrates the benefits of OOP in managing complex software systems and improving code readability and organization.

The game features the following functionality:

1. **Player Interaction:** Players take turns entering their moves by specifying the row and column where they want to place their mark.
2. **Board Display:** The current state of the game board is displayed after each move, allowing players to see the results of their actions.
3. **Win and Draw Detection:** The game checks for a winner after each move. If a player has succeeded in placing three marks in a row, the game announces the winner. If the board is full and no player has won, the game declares a draw.
4. **Replay Option:** After a game concludes, players have the option to start a new game without restarting the program.

By implementing these features using OOP techniques, the project not only provides a functional game but also serves as an educational example of how OOP can be applied to real-world problems in software development. This report will detail the design and implementation process, providing insights into the decisions made and the benefits of using OOP in this context.

UML-



MAIN Class

TicTacToe

MEMBERS

- `TicTacToe()`: Constructor that initializes the board and sets the starting player to 'X'.
- `printBoard()`: Displays the current state of the game board.
- `placeMark(row: int, col: int): bool`: Places the current player's mark on the board at the specified position if it's valid.
- `checkWinner(): bool`: Checks if the current player has won the game.
- `isBoardFull(): bool`: Checks if the game board is full.
- `switchPlayer(): void`: Switches the current player between 'X' and 'O'.
- `playGame(): void`: Controls the main game loop, handling player input, checking for a win or draw, and prompting to play again.

Attributes:

- `board: char[3][3]`: A 2D array representing the game board.
- `currentPlayer: char`: A character representing the current player ('X' or 'O').

ONLINE QUIZ GAME

INTRODUCTION:

This report presents the development of a Math Quiz Game implemented in C++ using Object-Oriented Programming (OOP) principles and templates. The game is designed to provide an engaging and educational experience for players of varying skill levels, offering easy, medium, and hard quizzes.

Key features of the game include:

- **Interactive Gameplay**: Players choose their difficulty level and answer a series of math-related questions. They receive immediate feedback on their answers, enhancing their learning experience.
- **Dynamic Question Handling**: The game uses templates to handle different types of questions and answers, making it versatile and extendable.
- **User-Friendly Interface**: The game provides clear instructions, displays results, and offers the option to replay, ensuring an enjoyable user experience.
- **OOP Design**: The game's architecture leverages inheritance and polymorphism, encapsulating the game logic within a base class and specific behaviors within a derived class.

Enhancing output at console using functions:

Void SetColor(int Color)

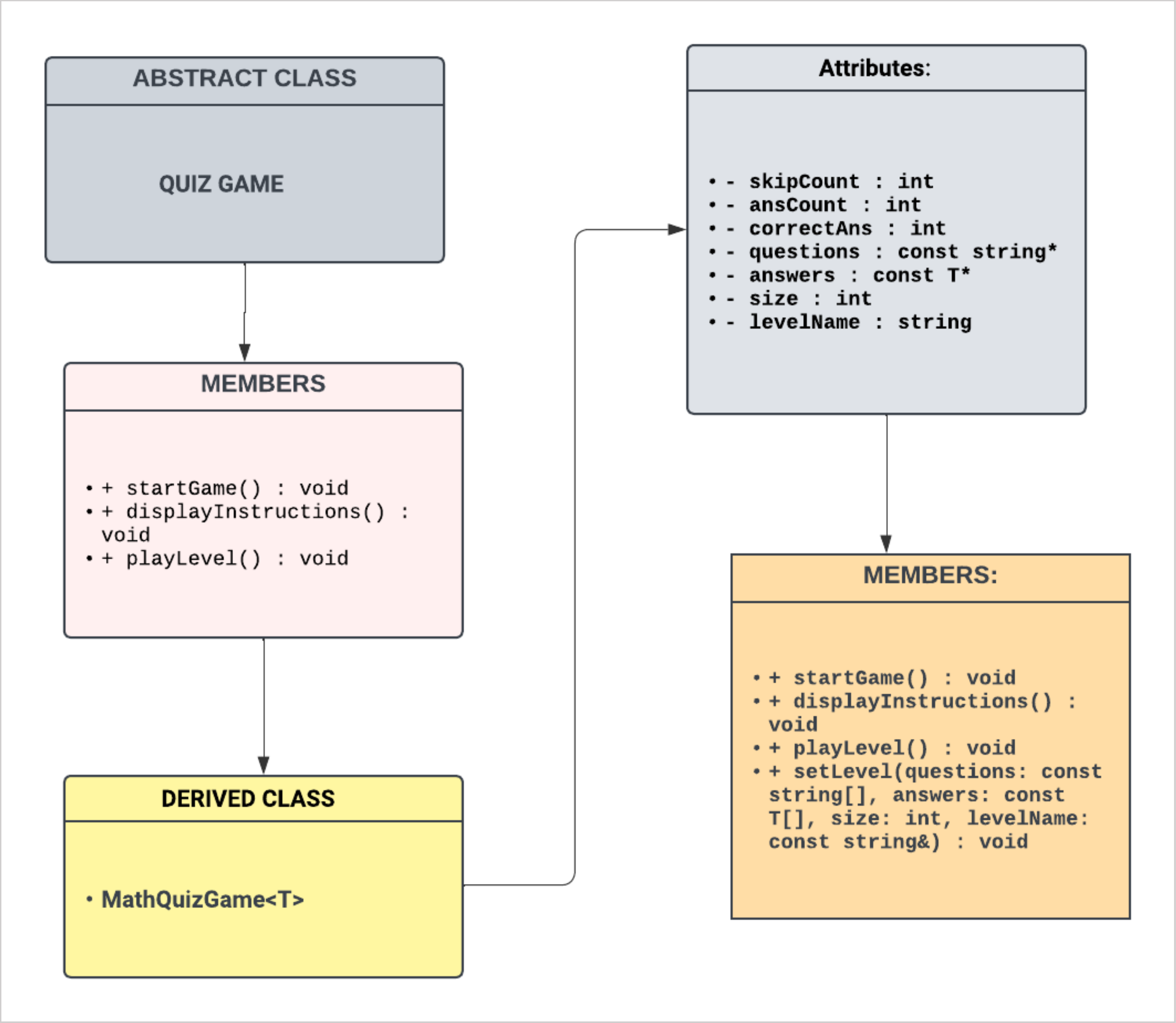
This Function sets the color of the text displayed in the console using windows API.

“int Color” is an integer value representing the color attributes.

Set Console Text Attribute(GetStd Handle (STD_OUTPUT_HANDLE),color);

This function sets the attributes of the console screen buffer ,meaning it changes the text and background color based on “color” parameter.

UML-


ABSTRACT CLASS
QUIZ GAME
MEMBERS
+ startGame() : void
+ displayInstructions() : void
+ playLevel() : void
DERIVED CLASS
MathQuizGame<T>
Attributes:
- skipCount : int
- ansCount : int
- correctAns : int
- questions : const string*
- answers : const T*
- size : int
- levelName : string
MEMBERS:
+ startGame() : void
+ displayInstructions() : void
+ playLevel() : void
+ setLevel(questions: const string[], answers: const T[], size: int, levelName: const string&) : void

LEARNING OUTCOMES:

Through the implementation of both the Tic-Tac-Toe and Quiz Game projects using Object-Oriented Programming (OOP) principles, we have learned and achieved the following:

- **Fundamental OOP Concepts**:
  We gained a solid understanding of encapsulation, inheritance, and polymorphism.
  We effectively utilized access specifiers (**private**, **protected**, **public**) to manage data access and maintain data integrity.

- **Class Design and Implementation**:
  We designed and implemented classes to model game boards, players, and game logic.
  This experience improved our ability to structure and organize code using constructors, member functions, and member variables.

- **Templates and Dynamic Memory**:
  We utilized templates to handle different data types in the Math Quiz Game, enhancing code flexibility and reusability.
  Dynamic memory management was applied for arrays of questions and answers, ensuring efficient memory usage.

- **User Interface and Input Handling**:
  We developed interactive user interfaces using console input and output functions.
  Input validation techniques were implemented to handle and correct user errors effectively.

- **Game Logic and Algorithms**:
  Implemented game-specific logic and algorithms for both Tic-Tac-Toe and the Math Quiz Game.
  We refined our algorithms through iterative development and testing, ensuring robust handling of edge cases.

- **Software Development Process**:
  We adopted an iterative development approach, testing each part of our programs incrementally.
  Code debugging and refinement improved performance, readability, and maintainability.
- **Console Output Formatting**:

Utilized console functions for formatting output, including text color changes and screen clearing.
Dynamic content, such as displaying the current date and time, enhanced user interaction.

- **Object-Oriented Analysis and Design (OOAD)**:
  Applied OOAD principles to analyze requirements, design class structures, and implement game features.
  This practice led to the development of modular and scalable software systems.

- **Reusability and Extensibility**:
  Designed games with OOP principles to ensure code reusability and facilitate easy feature addition.
  This approach supports future modifications and enhancements.

- **Team Collaboration and Project Management**:
  Collaborated effectively within a team, dividing tasks and managing code integration.
  Peer reviews and teamwork improved project efficiency and code quality.
  These projects have provided practical experience in software development, preparing us for future endeavors in computer programming and software engineering.